

Reconfigurable Meshes: Theory and Practice¹

Kiran Bondalapati and Viktor K. Prasanna
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-2562

Abstract

Configurable computing has recently gained much attention with the promise of delivering an order of magnitude performance improvement over general purpose processors. In this paper we contrast the abstract models of reconfigurable architectures and actual hardware available for configurable computing systems.

There is a wealth of ideas related to abstract models of reconfigurable architectures and fast parallel algorithms which exploit the reconfiguration potential in non-trivial ways. We summarize these abstract models and illustrate the power of these models using several example algorithms. We identify the practical problems in implementing these models in VLSI and describe some prototype implementations. Commercial FPGA devices which are being touted as the solution for building configurable computing systems are also examined. The MAARC² project at USC endeavors to bridge this gap between the abstract and the real worlds.

¹This work was supported by DARPA under contract DABT63-96-C-0049 monitored by Fort Haachuca.

²Models, Algorithms and Architectures for Reconfigurable Computing, <http://maarc.usc.edu>

1 Introduction

Configurable computing has recently gained much attention with the promise of delivering an order of magnitude performance improvement over general purpose processors. The paradigm of computing in space, i.e., laying out a series of computations on several functional units, as opposed to computing in time, i.e., a series of computations executed in sequence on a single functional unit, is being actively explored. There are several directions in which research is being carried out to realize the potential of configurable computing.

The idea of a VLSI array of processors overlaid with a reconfigurable bus system and an abstract model based on this architecture was proposed in [23]. Several abstract models of reconfigurable architectures and fast parallel algorithms for many problems have been described in the literature. These models include the bus automaton [30], content addressable array parallel processor (CAAPP) [33], polymorphic processor array (PPA) [21], among others. Efficient algorithms for fundamental data movement operations [23, 24], sorting [2, 11, 27, 28], arithmetic [15, 29], graph problems [24], image processing [14, 16] and computational geometry [12] have been developed on reconfigurable meshes. There have been several research prototype implementations of reconfigurable architectures which are related to the abstract models. Such architectures include the GCN [33], YUPPIE [22], CLIP [10], PADDI [7], ABACUS [5], DPGA [8].

Currently the architectures which are being utilized to design reconfigurable systems have their root in Field Programmable Gate Array (FPGA). FPGAs consist of a matrix of fine grain computational elements, usually implemented using lookup tables, with a hierarchy of programmable interconnect. Traditionally, FPGAs have been used for logic design and hardware emulation. Their suitability as computing engines for reconfigurable architectures is being explored in SPLASH [6], DEC PeRLe [3], Teramac [1], among others. But FPGA architectures have been primarily designed to emulate random logic without frequent reconfiguration. Also, on-chip memory capacities are too small, reconfiguration times are relatively long (several milliseconds) and partial reconfiguration is difficult.

The advent of static RAM based FPGA devices has given rise to new opportunities in reconfigurable computing area. These devices provide features which allow changing the device configuration on the fly. But reconfiguration cost is still the prohibitive factor in using them for configurable computing. The other major factor is the lack of software tools which allow synthesis of applications exploiting dynamic reconfiguration. Research is also being carried out in designing coarser grain architectures which incorporate reconfigurable features such as MATRIX [25], BRASS Garp [36], RaPiD [9], CMU CVH [37], COLT [4].

This paper looks at the two extremes of the configurable computing world, the abstract models and actual devices. Though the abstract models have been shown to be very powerful, they are difficult to realize in VLSI. There have been several research prototypes of devices that show promise of implementing reconfigurability. But configurable computing cannot deliver the promise until commercial devices strive to deliver the reconfiguration potential possible with current VLSI technology.

In Section 2 we describe and characterize several variants of the reconfigurable mesh model. In Section 3 we illustrate the power of reconfiguration by describing algorithms for EXOR, Addition, Sorting, Prefix operations and Component labeling. We examine the technical issues in implementing these models and give brief descriptions of several implementations in Section 4. Some commercial devices which look promising for designing configurable systems are also explored in this section. Concluding remarks are made in Section 5.

Figure 1: Reconfigurable Mesh.

The basic computational unit of the reconfigurable mesh is the Processing element (PE) which consists of a switch, local storage and an ALU (Fig. 1). In a unit time, a PE can perform:

1. Setting up of a connection pattern.
2. Read from or write onto a bus or local storage.
3. Logical or arithmetic operations on local data.

Various models of reconfigurable meshes have been proposed in the literature. Most of these models are synchronous in nature and permit unconditional global switch setting in addition to local switch control. Unconditional global switch setting is performed by the broadcast of a global instruction from a central controller. Reconfigurable mesh models can be characterized by the following parameters:

- **Width** It refers to the data width of the PE. The two classes of models which have been proposed are bit and word models. The main difference is the width of the input operands of the PE. Also, $\log n$ bits (where n is the size of the reconfigurable mesh) need to be accessed when the processor needs to know its position before setting its configuration. Note that the **Width** parameter is not directly related to the bus width of the reconfigurable mesh.
- **Delay** One critical factor in the analysis of reconfigurable algorithms is the time needed to propagate a signal. Some models assume this to be a unit-time operation no matter how far the signal has to travel, while other models assume this to be a function of the number of processors. Time analyses which assume constant time are called *unit-delay models* and logarithmic time are called *logarithmic-delay models*.
- **Bus Access** Each PE connects to the bus through its ports and will either read or write to it. Similar to shared memory machines the models can be classified as CRCW, CREW, ERCW, and EREW based on how the bus is accessed. The most common models are the ERCW models but the CRCW models have also been extensively studied. The CRCW models typically assume that a wired-or operation is performed on a concurrent write by multiple PEs onto the bus.
- **Connection Patterns** Each PE can set the connection between its four ports based on local data or global instruction. There are a total of 15 different connection patterns possible. Different models differ in the number of connection patterns (a subset of 15) which they allow. These models can also be classified based on whether they allow cross-over of the port connections. The models which allow cross-over of connections (such as N-S and E-W) have been shown to be more powerful than the non-cross-over models.

2.1 Various Models

Since the introduction of the reconfigurable mesh [23], several models have appeared in the literature. Following variations of the models have been studied extensively and efficient algorithms have been developed for several problems:

- **PARBS** The most general and the most powerful is the **PARBS** model [32]. In this model no restriction is placed on the allowed connections among the 4 I/O ports in each PE. Thus, all 15 connection patterns are possible and algorithms for a variety of applications have been developed on this model [11, 14, 29, 32].
- **RMESH** This consists of two-dimensional mesh of size $n \times n$, with each PE connected to a broadcast bus [23]. This bus, like the mesh, is also constructed as an $n \times n$ grid, where PEs are located at the intersection of the grid lines. Further each bus link between adjacent PEs has a switch embedded in it, where the two PEs at either end of the link can control the switch. When all the switches are closed, all the n^2 PEs are connected together. If all the PEs disconnect the switches to the north, then we obtain row buses. Similarly column buses can be obtained. The connection patterns allowed in RMESH are shown in Figure 2.
- **MRN/LRN** The Reconfigurable Network (RN) [2] is a general model in which no restriction is placed on the bus segments that connect the PE or on the placement of the

Figure 3: Connection patterns allowed in MRN.

- **Polymorphic Torus** A polymorphic torus architecture [18, 22] is identical to the PARBS architecture except that the rows and columns of the underlying mesh wrap around.

Jang *et. al.* proposed a **Bit Model** [13] of reconfigurable mesh which can simulate (asymptotically) most of the word based models of the reconfigurable mesh in the same amount of

time using the same VLSI area. The basic PE in the Bit Model consists of a switch, local storage and a 1-bit ALU. The switch consists of six bit-level switches which can be closed or opened using local information within the PEs. The switch can realize any of the possible 15 connection patterns among its 4 I/O ports. The bus architecture is similar to the RMESH architecture and can carry $O(1)$ bits of data.

2.2 Related Models

- **REBSIS**

In the reconfigurable buses with shift switching (REBSIS) [20] model, each word level switch consists of several bit level switches. In each connection pattern each of the bit level switches share a common connection pattern to control the bit level buses in a uniform way. Based on a control bit pattern the switch performs rotate-shift on the input bit pattern. It has been proved that that the REBSIS model is more powerful than several word models [20] but the Bit Model [13] of reconfigurable mesh has been shown to be able to simulate the REBSIS model using the same area.

- **RMBM** A more general reconfigurable network model called the reconfigurable multiple bus machine (RMBM) [31] was proposed to investigate effects of switch models on relative computational power of reconfigurable network models. This model separates the computational aspects from the connection configuration aspects. The RMBM model has processors, buses, fuse lines and sets of switches. Each processor has one write port and several read ports. The switches can be classified into connect switches, segment switches and fuse switches. The connect switches connect a particular port of a processor to one of the buses, the segment switches segment the bus and the fuse switches connect two or more buses together. There are restricted versions of this model which differ in the classes of switches which they allow.

3 Some Illustrative Algorithms

Lot of work has been done in exploiting the power of reconfigurable meshes. Algorithms for basic computations such as Or, And, Exor, Addition, Multiplication etc. have been designed and shown to be optimal on several variants of the reconfigurable mesh models. Using these basic data operations and additional non-trivial techniques of exploiting reconfiguration, algorithms for problems in image processing, computational geometry, graphs etc. have been designed. In this section some algorithms are described to illustrate the power of these architectures.

3.1 EXOR Computation

The EXOR of N bits of data can be computed on a reconfigurable mesh of size $2n \times 3$ in $\theta(1)$ time using the unit-time delay model and in $\theta(\log n)$ time using the log-time delay model [24]. The basic idea behind the algorithm is described here.

Based on a single input bit a 3×2 array of PEs set their local switch configurations to one of the two patterns as shown in Figure 4. If the input bit is 1 the top two rows cross-over and the 1-signal toggles to the other row and if the input bit is 0 then the 1-signal passes through

Figure 4: EXOR computation

3.2 Addition

Addition of two n -bit numbers can be carried out in a similar way as EXOR computation. Each PE sets its switch pattern based on either a carry generate or a carry propagate configuration. If the two input bits a_i and b_i are different then the PE connects its West input to the East output port, which is a carry propagate configuration. If the input bits are the same then none of the switches are connected. The carry generate at a PE is implemented by the PE writing a 1 on its East port when both the bits a_i and b_i are 1.

An example addition of two 5-bit numbers is shown in Figure 5. The bits c_i indicate the intermediate carry bits and z_i are the result bits.

Using a similar idea and constructing a k -stage ripple carry adder it was shown that addition of n k -bit numbers ($1 \leq k \leq n$) can be performed in constant time using a $n \times nk$ bit model of reconfigurable mesh [11].

3.3 Parallel Prefix

Parallel Prefix is an important operation that can be used to sum values, broadcast data, solve problems in image processing and graph problems etc. [24]. Assume processor p_i , $0 \leq i \leq n-1$, initially contains the data element a_i . The parallel prefix problem requires p_i to compute $a_0 \otimes a_1 \otimes \dots \otimes a_i$, where \otimes is an associative operator such as addition (+).

Figure 5: Addition of two 5-bit numbers.

The given n values are assumed to be distributed one per processor on a reconfigurable mesh of size n . The binary associative operation \otimes , is assumed to be a unit-time operation. First, parallel prefix is performed along the rows so that each processor knows the initial prefix of those values restricted to its row. Next, in the last column the parallel prefix is performed to determine row-wise prefix solutions. Finally, within each row, the prefix of previous of the previous rows is broadcast so that all the processors can update their entry appropriately. Parallel prefix can be computed in every row simultaneously in $\log n^{1/2}$ iterations by appropriately setting switches, broadcasting and updating values at each iteration.

3.4 Sorting

There are several sorting algorithms on reconfigurable mesh models. We describe here the algorithm presented in [11]. Sorting of a sequence can be decomposed into sort of its subsequences and data movement between the sorted subsequences. The reconfigurable mesh algorithm uses a variation of Leighton's eight-stage column sort [17]. The stages are a combination of stages of $n^{1/4}$ sorters, each capable of sorting $n^{3/4}$ numbers, and $n^{1/4}$ -shuffle network stages.

The input sequence of n numbers is assumed to be initially stored in the top row of the reconfigurable mesh. The sequence is partitioned into subsequences of $n^{3/4}$ numbers each. Sorting of a subsequence is done by computing the ranks of all the numbers and then storing each number according to its rank by using shuffle networks [11]. Sorting of $n^{3/4}$ numbers in constant time is carried out using a $n \times n^{3/4}$ reconfigurable mesh. In the first step, each of the $n^{3/4}$ PEs broadcast their numbers along each column of n PEs. Then the mesh is divided into $n^{3/4}$ submeshes each of size $n^{1/4} \times n^{3/4}$. The rank of number x_i is computed by submesh i using row broadcasts. The results of the comparisons made after this row broadcasts are added to give the rank of each number. The addition can be done in constant time as stated in Section 3.2. The $n^{1/4}$ -shuffle stage can also be implemented in constant time using a sequence of broadcast operations.

3.5 Component Labeling

The problem is to label the connected components of a digitized image. Given an $n \times n$ image which is distributed as a pixel per processor onto the processors of a reconfigurable mesh of size $n \times n$, the connected components can be labeled in $\theta(\log n)$ time under the unit-time delay model [24].

In the first step each processor examines the pixels in each of its four neighbors and

sets its four switches so that a connection is maintained only between neighboring black pixels. This $\theta(1)$ operation creates a subbus over each component. Given a linked list of processors overlaid by a reconfigurable subbus, the minimum(maximum) of the value stored in these processors can be computed in $O(\log n)$ iterations. Each iteration computes the local minima(maxima) and discards the other elements. Each iteration uses a constant number of broadcast steps and comparison operations, and hence the total running time is as stated above.

3.6 Summary of Results

We present a brief summary of algorithms on the reconfigurable mesh models. A comprehensive bibliography of results can be found in [26]. All results are with respect to the unit-time delay reconfigurable mesh model.

Problem	Mesh Size	Time
EXOR of n bits	$2n \times 3^*$	Constant
Prefix-And of n 1-bit numbers	$1 \times n^*$	Constant
Maximum(Minimum) of $n \log n$ -bit numbers	$n \times n$	Constant
Addition of n k -bit numbers, $1 \leq k \leq n$	$n \times nk^*$	Constant
Multiplication of two n -bit numbers	$n \times n^*$	Constant
Division of two n -bit numbers	$n \times n^*$	Constant
Histogram of an $n \times n$ image (h gray levels)	$n \times n$	$O(\min(\sqrt{h} + \log(\frac{n}{h}), n))$
Sort of n $O(\log n)$ bit numbers	$n \times n$	Constant
Convex Hull of n points	$n \times n$	Constant
Smallest enclosing rectangle of n points	$n \times n$	Constant
Triangulation of n planar points	$n^2 \times n$	Constant
All-pairs nearest neighbors of n points	$n \times n$	Constant
Two-set dominance counting of n points	$n \times n$	Constant
Connected components of an $n \times n$ image	$n \times n$	$O(\log n)$

* - the bit model of reconfigurable mesh is used.

4 Practical Considerations and Architectures

The choice of an architecture is strongly influenced by physical fabrication constraints. The reconfigurable mesh has nearly constant diameter and a dynamically reconfigurable bus system. It is very attractive in terms of implementation because of the two dimensional topology, low pin requirement and highly regular structure, which are well suited for today's VLSI and packaging technology.

There are several physical constraints that have to be overcome to successfully implement these architectures. Some of the features of the reconfigurable mesh models which should be examined in the context of hardware technology are:

- **Reconfiguration** The ability to set the local configurations of switches is one of the key aspects of reconfigurable meshes which is exploited in designing efficient algorithms. Assumptions made in the model impact the design since more flexibility is allowed

switch patterns usually implies more area because of larger control memory etc. Most implementations support global control signals but implementing dynamic change of configuration based on local data is very expensive and is difficult to provide in general purpose implementations.

- **Signal Delay** There is potentially a large signal delay due to a long chain of shorted path, set up because of configuration. The signal propagation time grows linearly with the length of the wire carrying the signal. There are also unpredictable delays in VLSI because the wire capacitance is affected by the number of processors connected to the wire carrying the signal.

Recent VLSI implementations have addressed these issues and suggest that the broadcast delay, although not constant, is very small. For example, only 16 machine cycles are required to broadcast on a 10^6 processor YUPPIE. GCN has shorter delays by adopting all-active and pre-charged circuit for local switches. ABACUS architecture propagates a signal through 18 PEs in a single 8ns clock cycle. Broadcast delay can be further reduced by using optical fibers for reconfigurable bus system and using electrically controlled directional coupler switches for connecting and disconnecting two fibers.

- **Clock Timing** In reconfigurable meshes variable length shorted path can be established based on the algorithm. If a fixed length clock is designed to accommodate the worst case shorted path, the clock for the system will be degraded. The constant time algorithms in the literature do not consider the clock implementation. Many clocking schemes are possible to accommodate the worst case path while not affecting the average clock performance. One such proposal is variable length clock that adjusts the length of clock to the length of the path. Global distribution of control signals also affects the clock signals. Detailed discussion of such issues is beyond the scope of this paper.

4.1 Architectures

We look at two variants of reconfigurable architectures. One class of architectures are based on the abstract models and try to approximate the features of the models. We describe the YUPPIE and the ABACUS architectures which are representative research prototypes. The other class consists of architectures which have evolved from commercial FPGA designs. We look at the features offered by two FPGAs, namely, XILINX 6200 and the NSC CLAY. Though these devices have not been designed for reconfigurable computing engines, they are a result of demand for fast reconfigurable components.

4.1.1 Polymorphic Torus Architecture - YUPPIE

The Polymorphic Torus [22] consists of a physical network (PNET) and a programmable internal network (INET) at each node of the PNET. The PNET is global while the INET is local. In a Polymorphic Torus consisting of $n \times n$ processors, the PNET is an $n \times n$ mesh with its boundary connected in either torus mode or spiral mode. Except for selection of torus or spiral mode, the PNET is a hard-wired, fixed, non-programmable network. In contrast INET is totally programmable. Each of the four ports of the INET can be connected to any port.

The VLSI implementation of the 2D Polymorphic Torus is called YUPPIE (Yorktown Ultra Parallel Polymorphic Image Engine). YUPPIE follows a regular SIMD model of com-

putation with a central controller (CC) generating a stream of instructions. A processor array (PA), made up of many bit-serial PEs connected by a Polymorphic Torus receives the instruction stream from the CC and executes it. PEs can be selectively disabled based on local condition, but all enabled PEs carry out the same operations on their own data. Data Memory (DM) consists of on-chip 256 fast-access one-bit registers for each PE, termed local data memory (LDM), and off-chip external data memory (EDM). The YUPPIE chip consists of 16 nodes arranged as a 4×4 mesh. A programmable length clock generator (PLCG) generates the timing signals for YUPPIE, since it needs to be driven by a variable length clock.

The YUPPIE PE has a 1-bit ALU, carry register (CY), data registers (A, TR) and two control registers (EN, CCR). All registers are 1-bit wide and one of the data registers functions as the accumulator. The ALU can carry out basic addition and boolean operations with operands from physical links, local or external memory and/or the data registers. The INET switching is established by choosing one of two patterns broadcast by the CC. This choice is made depending on the data in one of the control registers, namely, CCR.

Implementation using a 2 micron CMOS technology with two metal layers has shown a less than 20% overhead for the programmable interconnect and the ability to propagate the signal through 16 PEs in a single clock cycle.

4.1.2 ABACUS

ABACUS [5] is a distributed bit-parallel (DBP) architecture based on the reconfigurable mesh. The ABACUS processing element (PE) contains 64 bits of dual-ported memory in two banks and two 3-input ALUs, each of which takes two inputs from its memory bank and one input from the other bank. Part of the memory bank is utilized as control registers for enabling PE and network operation.

At each PE, the network is composed of a wired-OR bus and four isolating switches. When all switches are open, two network control bits specify which of the four nearest neighbors is connected to the PE input. Each PE can also close the switch in the read direction, the other three switches remain open unless closed by a neighboring PE. Connected processors form a multiple-writer wired-OR bus. VLSI implementation of the network consists of a pre-charged bus which is pulled down by any PE writing a one. Additional delays are reduced by using local accelerator circuits.

There are additional circuits for reading and writing to on-chip distributed memory and interface circuitry to external data memory. A VLSI implementation in 0.8/1 micron technology is expected to sustain a 8 ns cycle time. Simulations show that in worst case the signal can propagate through 18 PEs in a single clock cycle. A single ABACUS IC is expected to deliver 1-5 giga-operations per second (GOPS) on 16-bit arithmetic operations. This is approximately 20 to 100 times faster than microprocessors implemented in comparable VLSI technology.

4.1.3 XILINX XC6200

The XC6200 FPGA [41] architecture from Xilinx is the first SRAM based FPGA architecture designed for implementing reconfigurable coprocessors. The XC6200 architecture features a fine-grained cell structure, abundant routing, built-in processor interface and supports fast partial reconfiguration.

The programmable logic of an XC6200 consists of large array of reconfigurable logic

cells each of which contains both programmable logic and routing resources. Each cell contains a flip-flop and combinatorial logic capable of implementing any two-input function or any type of 2-to-1 multiplexer. Cells are arranged in 4-by-4 blocks and 16-by-16 tiles. The interconnection network consists of a hierarchy of programmable routing wires. Each cell can be used for logic or memory functions. When cells are configured as memory, each cell provides two bytes of ROM or RAM memory which can be accessed externally or internally.

The most important feature in the new XC6200 device is the FastMap interface, designed to connect directly to an external processor's system bus. The FastMap interface places the whole FPGA into the processor's address space. The processor can read and write the logic and the configuration memory by using normal load and store. This parallel interface allows the entire configuration memory to be programmed in under 100 micro-seconds.

A random access feature allows arbitrary areas of the FPGA memory to be changed. This provides a fast partial reconfiguration capability. This partial reconfiguration can be performed without disturbing circuits running in other parts of the device. This facilitates sharing of hardware space by swapping in and out designs at runtime. A reconfigurable hardware platform based on the XC6200 architecture has been designed and is being offered as a commercial product by Virtual Computer Corporation [40].

4.1.4 NSC CLAy

The National Semiconductor CLAy [39] architecture is an SRAM based Configurable Logic Array. CLAy was designed to support real-time algorithm and logic sharing by using dynamic partial reconfiguration.

The logic cell layout is similar to existing FPGA devices, with a flip-flop and 5-input lookup tables. The interconnection network is made up of nearest neighbor connections, local and express bus wires. The full device can be configured in 640 micro-seconds. Larger designs are supported by an integrated Field Configurable Multi-Chip Module (FCMCM) which consists of a 2×2 array of CLAy devices.

CLAy supports partial reconfiguration by which a single cell's functionality can be changed. This is much faster than programming the complete device and reconfiguration time is the order of 1 micro-second. This partial reconfiguration can be done without functional interruption of the remaining parts of the device. These features of the CLAy devices have been exploited in designing novel applications [35].

5 Conclusion

Configurable computing holds lot of promise for the future. To realize this potential we need a variety of system architectures, algorithmic techniques and software tools. We have discussed the abstract models of reconfigurable architectures and algorithms using these models. The technology constraints in realizing these architectures have been examined and prototype implementations which try to overcome these constraints have been described.

Currently, the designs of configurable computing systems are based on fine-grained commercial devices like Field Programmable Gate Arrays. Since FPGAs were primarily designed for logic emulation there has not been much progress in trying to achieve fast reconfiguration times. Recent SRAM based devices have started addressing this issue and we examine some of these devices. We believe that the wealth of ideas in the abstract world can be leveraged in

designing systems and algorithms which exploit the available reconfiguration potential. The MAARC [38] project at USC explores these opportunities.

6 Acknowledgment

Over the years, several students at USC have conducted research in reconfigurable architectures and algorithms. We would like to acknowledge the contributions of Ju-wook Jang, Heonchul Park and Dionisis Reisis. This paper summarizes many of their contributions in reconfigurable meshes and algorithms.

References

- [1] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes and G. Snider, "Teramac - Configurable Custom Computing", *IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 32-38, April 1995.
- [2] Y. Ben-Asher, D. Peleg, R. Ramaswami and A. Schuster, "The Power of Reconfiguration", *Journal of Parallel and Distributed Computing*, V. 13, No. 2, pp. 139-153, 1991.
- [3] P. Bertin, D. Roncin and J. Vuillemin, "Programmable active memories: a performance assessment", In F. Meyer auf der Heide, B. Monien, and A. L. Rosenberg, editors, *Parallel Architectures and their efficient use*, pp. 119-130, Lecture notes in Computer Science, Springer-Verlag, October 1992.
- [4] R. Bittner and P. Athanas, "Wormhole Run-time Reconfiguration", *ACM International Symposium on Field Programmable Gate Arrays*, pp. 79-85, February 1997.
- [5] M. Bolotski et. al., "Abacus: A High-Performance Architecture for Vision", *International Conference on Pattern Recognition*, 1994.
- [6] D. A. Buell, J. M. Arnold and W. J. Kleinfelder, "Splash 2: FPGAs in a Custom Computing Machine", IEEE Computer Society Press, 1996.
- [7] D. Chen and J. Rabaey, "A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real Time Data Paths", *ISSCC*, Feb. 1992.
- [8] A. DeHon, "DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century", *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1994.
- [9] C. Ebeling, D. C. Cronquist, P. Franklin. "RaPiD - Reconfigurable Pipelined Datapath", *6th International Workshop on Field-Programmable Logic and Applications*, 1996.
- [10] T. J. Fountain and V. Goetcherian, "CLIP4 Parallel Processing System", *Proc. of the Institute of Electrical Engineers*, Vol. 127E, pp. 219-224.
- [11] J. Jang and V. K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh," *Journal of Parallel and Distributed Computing*, pp. 31-41, Vol. 25, 1995.
- [12] J. Jang, M. Nigam, V. K. Prasanna and S. Sahni, "Constant Time Algorithms for Computational Geometry on the Reconfigurable Mesh," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1-12, Vol. 8, January 1997.

- [13] J. Jang, H. Park and V. K. Prasanna, "A Bit Model of Reconfigurable Mesh", *Proc. of Reconfigurable Architectures Workshop: International Parallel Processing Symposium*, April 1994.
- [14] J. Jang, H. Park and V. K. Prasanna, "A Fast Algorithm for Computing the Histogram on Reconfigurable Mesh," *Proc. of Frontiers of Massively Parallel Computation*, pp. 244-251, October 1992.
- [15] J. Jang, H. Park and V. K. Prasanna, "An Optimal Multiplication Algorithm on Reconfigurable Mesh," *Proc. of Symposium on Parallel and Distributed Processing*, pp. 384-391, December 1992.
- [16] J. Jenq and S. Sahni "Reconfigurable Mesh Algorithms For Image Shrinking, Expanding, Clustering, and Template Matching", *Proc. of Fifth International Parallel Processing Symposium*, pp. 208-215, April 1991.
- [17] F. T. Leighton, "Tight bounds on complexity of parallel sorting", *IEEE Transactions on Computers*, Vol. 34, pp. 344-354, 1985.
- [18] H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. on Computers*, Vol. 38, No. 9, pp. 1345-1351, Sept. 1989.
- [19] H. Li and Q. F. Stout, "Reconfigurable SIMD Massively Parallel Computers", *Proceedings of the IEEE*, Vol. 79, No. 4, pp. 429-443, April 1991.
- [20] R. Lin and S. Olariu, "Reconfigurable Buses with Shift Switching: Concepts and Applications", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 1, pp. 93-102, January 1995.
- [21] M. Maresca, "Polymorphic Processor Arrays", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 5, pp. 490-506, May 1993.
- [22] M. Maresca and H. Li, "Connection Autonomy in SIMD Computers: A VLSI Implementation", *Journal of Parallel and Distributed Computing*, Vol. 7, pp 302-320, 1989.
- [23] R. Miller, V. K. Prasanna Kumar, D. I. Reisis and Q. F. Stout, "Meshes with Reconfigurable Buses", *Proc. 5th MIT Conference On Advanced Research in VLSI*, pp. 163-178, March 1988.
- [24] R. Miller, V. K. Prasanna Kumar, D. I. Reisis and Q. F. Stout, "Parallel Computations on Reconfigurable Meshes", *IEEE Trans. on Computers*, July 1993.
- [25] E. Mirsky and A. Dehon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources", *IEEE Symposium on FPGAs for Custom Computing Machines*, April 1996.
- [26] K. Nakano, "A Bibliography of Published Papers on Dynamically Reconfigurable Architectures, *Parallel Processing Letters, Special Issue on Dynamically Reconfigurable Architectures*, 1995.
- [27] K. Nakano, T. Msuzawa, N. Tokura, "A Fast Sorting Algorithm on a Reconfigurable Array", *The Institute of Electronics, Information and Communication Engineers*(Japanese), COMP 90-69, December 1990.

- [28] M. Nigam and S. Sahni, "Sorting n numbers on $n \times n$ Reconfigurable Meshes with Buses", International Parallel Processing Symposium, pp. 174-181, April 1993.
- [29] H. Park, V. K. Prasanna and J. Jang, "Fast Arithmetic on Reconfigurable Meshes", International Conference on Parallel Processing, August 1993.
- [30] J. Rothstein, "Bus automata, brains, and mental models", *IEEE Transactions on Systems Man Cybernetics*, Vol. 18, pp. 522-531, 1988.
- [31] R. K. Thiruchelvan, J. L. Trahan, and R. Vaidyanathan, "On the Power of Segmenting and Fusing Buses", *Proc. of International Parallel Processing Symposium*, pp. 79-83, April 1993.
- [32] B. F. Wang and G.H. Chen, "Constant Time Algorithms for the Transitive Closure Problem and Some Related Graph Problems with Reconfigurable Bus Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 4, pp. 500-507, 1991.
- [33] C. C. Weems, "The Image Understanding Architecture," *International Journal of Computer Vision*, V. 2, pp. 251-282, 1989.
- [34] C. C. Weems and J. H. Burrill, "The Image Understanding Architecture and its Programming Environment," *Parallel Architectures and Algorithms for Image Understanding*, V. K. Prasanna Kumar(Ed.), Academic Press, 1991.
- [35] M. J. Wirthlin and B. L. Hutchings, "Improving Functional Density Through Run-Time Constant Propagation", *ACM International Symposium on Field Programmable Gate Arrays*, pp. 86-92, February 1997.
- [36] BRASS Homepage, <http://www.cs.berkeley.edu/projects/brass>.
- [37] CMU Cached Virtual Hardware Homepage, <http://www.ece.cmu.edu/afs/ece/usr/herman/www/CVH/intro/intro.html>.
- [38] MAARC Homepage, <http://maarc.usc.edu>.
- [39] National Semiconductor, *Configurable Logic Array (CLAy) Data Sheet*, December 1993.
- [40] Virtual Computer Corporation, "Reconfigurable Computing Products", <http://www.vcc.com/products/pci6200.html>.
- [41] Xilinx, *XC6200 Field Programmable Gate Arrays*, 1996.